

the
original
Hacker

creado por EUGENIA BAHIT

jogando con la inteligencia



SHANA TOVA

EDICION ESPECIAL

**50 TIPS, TRICKS & HACKS EN PHP
Y HOWTO PARA EUROPIO ENGINE**

©2014 Eugenia Bahit - GNU Free Documentation License

¿Conoces la diferencia entre un tip, un trick y un hack?

- **Tip:** es una palabra inglesa que significa «consejo». Un tip es una **recomendación** para realizar determinada acción obteniendo el mejor resultado posible.
- **Trick:** en castellano, significa «truco». Un truco es una **forma avanzada de resolver un determinado problema**, empleando elementos del lenguaje y técnicas de programación, específicos para la resolución del problema.
- **Hack:** el hack es muy fácil de confundir con el truco. Sin embargo, el hack es aquel que emplea soluciones laterales para la resolución del problema. Una **solución lateral**, es aquella que emplea recursos existentes con fines para los cuáles, dicho recurso, no ha sido pensado originalmente.

*Los **tips, tricks & hacks** de este compilado han sido extraídos en su mayoría de [Europio Engine](#).*

Tips, Tricks & Hacks en PHP

PARA SUS VERSIONES 5.3, 5.4 Y 5.5

© Copyright 2014

Eugenia Bahit

 [Sígueme](#)

Compilado de distribución Libre y Gratuita

www.originalhacker.org



© 2014 [Eugenia Bahit](#)

Registrado en [SafeCreative](#) (Nº de Registro: [1412232826294](#))

Bajo los términos de la licencia [GNU FDL](#)

(Free Documentation License)

Índice de Contenidos de la Edición Especial N°11 de The Original Hacker

por [Eugenia Bahit](#)

Refactoring

1. Desempaquetado de parámetros para sustituir el exceso de argumentos.....5
2. Funciones que solo definen variables temporales para evitar códigos espagueti.....6
3. ¿Defines muchas variables con el mismo valor dependiendo de una evaluación condicional o más?.....7

Buenas prácticas

4. Condicionales ternarios para evitar grandes bloques if/else.....9
5. Mostrar errores en desarrollo y ocultarlos en producción de forma dinámica10
6. Evitar errores con las rutas de importación al hacer un include o require.....10

Con una sola línea de código

7. Completar una contraseña de forma aleatoria hasta alcanzar la longitud esperada.....11
8. Todos los campos del array \$_POST a variables, limpios y saneados en un solo paso.....12
9. Importar todos los archivos de una carpeta12

Hacks Experimentales

10. Wrappers y decoradores como en Python.....13

Manipulación de Archivos

11. Mostrar archivos binarios desde directorios no accesibles por el navegador.....16
12. Mostrar imágenes contenidas en un archivo ZIP no accesible desde el navegador.....17

Programación Orientada a Objetos

13. Invocar métodos no estáticos de clases instanciables en el mismo paso, en versiones anteriores a 5.4.....18
14. Truco #13 aplicado a todas las clases (incluidas las built-in).....18
15. Llamar a cualquier función de PHP con estilo orientado a objetos.....19
16. Recuperación de objetos excluyendo propiedades innecesarias.....20
17. Guardar datos que no son propiedades del objeto.....21

Inteligencia Artificial

18. Obtener palabras claves de un bloque de texto.....22

MVC

19. Tip para manejar un sistema ABM mediante URL amigables implementando un pseudo patrón MVC sin emplear OOP.....23
20. Alternar entre menú de usuario y menú de administrador.....25

Bases de Datos

21. Ahorrar recursos reemplazando el uso de bases de datos por archivos.....26
22. Ordenar resultados de una consulta a base de datos sin usar SQL.....27
23. Filtrar resultados de una consulta a base de datos sin usar SQL.....28
24. Generación dinámica de queries.....29

Los «**Europa Engine HOWTO**» podrás encontrarlos a partir de **página 32**

#00:00 (truco) Refactoring

Desempaquetado de parámetros para sustituir el exceso de argumentos en funciones

Suele suceder que una función se nos llena de parámetros (argumentos) y resulta sumamente engorroso. Un ejemplo podría verse como el siguiente:

```
function foo($nombre, $apellido, $fecha_nacimiento, $nacionalida, $sexo, $pais,
    $ciudad, $cp) {
    print "$apellido, $nombre ($nacionalida)";
    # ...
}
```

Para evitarlo, el truco consiste en definir un único argumento y utilizar la función `extract()` para obtener todas las variables a partir de un array asociativo:

```
function foo($kwargs) {
    extract($kwargs);
    print "$apellido, $nombre ($nacionalida)";
    # ...
}
```

Al invocar a la función se le pasa un array asociativo:

```
$data = array(
    "nombre"=>"Juan",
    "apellido"=>"Pérez",
    "fecha_nacimiento"=>"23/04/2000",
    "nacionalidad"=>"uruguaya",
    "sexo"=>"M",
    "pais"=>"Argentina",
    "ciudad"=>"Lanús",
    "cp"=>1824
);

foo($data);
```

Las claves del array serán usadas como nombres de variables.

#01:00 (truco) Refactoring

Funciones que solo definen variables temporales para evitar códigos espagueti

Nuevamente, empleando `extract()` podemos evitar funciones kilométricas, aplicando una de las técnicas de *refactoring* más usadas: la extracción de código.

El truco consiste en retornar dichas variables sin necesidad de generar un *array* manualmente. Para ello, se crea una función que solo se encargue de definir las variables temporales retornándolas mediante `get_defined_vars()`:

```
function set_vars() {
    $nombre = "Juan";
    $apellido = "Pérez";
    $fecha_nacimiento = "11/11/2011";
    $nacionalida = "argentina";
    $sexo = "M";
    $pais = "Argentina";
    $ciudad = "Junín";
    $cp = "1111";
    return get_defined_vars();
}
```

En la función *refactorizada* (desde la cual se extrajo el código), se llama a la nueva función, extrayendo las variables que ésta retorna:

```
function refactorizada() {
    extract(set_vars());
    print $apellido . ', ' . $nombre;
    # ...
}
```

#02:00 (tip) Refactoring

¿Defines muchas variables con el mismo valor dependiendo de una evaluación condicional o más?

Si te encuentras en una situación similar a la siguiente:

```
if(condicion) {
    $uno = $variable * 45;
    $dos = file_get_contents('foo.php');
    $tres = str_replace(' ', '-', 'hola mundo');
    $cuatro = array(1, 2, 3);
} else {
    $uno = "datos insuficientes";
    $dos = "datos insuficientes";
    $tres = "datos insuficientes";
    $cuatro = "datos insuficientes";
}
```

La lógica indica que las variables deberían tener inicialmente el mismo valor y sobrescribirse en caso que la condición se cumpla:

```
$uno = "datos insuficientes";
$dos = "datos insuficientes";
$tres = "datos insuficientes";
$cuatro = "datos insuficientes";

if(condicion) {
    $uno = $variable * 45;
    $dos = file_get_contents('foo.php');
    $tres = str_replace(' ', '-', 'hola mundo');
    $cuatro = array(1, 2, 3);
}
```

Sin embargo, existe una mejor forma de cumplir con las consecuencias lógicas empleando mejores prácticas, como la asignación múltiple de variables:

```
list($uno, $dos, $tres, $cuatro) = array_fill(0, 4, "datos insuficientes");

if(condicion) {
    $uno = $variable * 45;
    $dos = file_get_contents('foo.php');
    $tres = str_replace(' ', '-', 'hola mundo');
    $cuatro = array(1, 2, 3);
}
```

Cuando el mismo escenario se presenta pero debiendo evaluar una condición para cada variable, otra buena alternativa suele ser la siguiente:

```
list($uno, $dos, $tres, $cuatro) = array_fill(0, 4, "datos insuficientes");  
  
if(condicion 1) $uno = $variable * 45;  
if(condicion 2) $dos = file_get_contents('foo.php');  
if(condicion 3) $tres = str_replace(' ', '-', 'hola mundo');  
if(condicion 4) $cuatro = array(1, 2, 3);
```


#03:00 (tip) Buenas prácticas

Condicionales ternarios para evitar grandes bloques if/else

Frente a un escenario en el cuál gran cantidad de variables se definirán según los resultados booleanos de una única evaluación condicional como en el siguiente caso:

```
if(condicion) {
    $variable = valor;
} else {
    $variable = otro valor;
}
```

Resulta más legible implementar condicionales ternarios:

```
$variable = (condicion) ? valor : otro valor;
```

Para saber: los condicionales ternarios no se limitan solo a la definición de variables. Pueden emplearse en conjunción con muchas otras instrucciones:

```
return (condicion) ? valor si verdadero : valor si falso;
print (condicion) ? valor si verdadero : valor si falso;
(condicion) ? accion si verdadero : accion si falso;
```

Un ejemplo con el antes y el después:

```
ANTES:
$bruto = 100;
if($bruto < 100) {
    $neto = $bruto;
} else {
    $neto = $bruto * 0.9;
}
```

```
DESPUÉS:
$bruto = 100;
$neto = ($bruto < 100) ? $bruto : $bruto * 0.90;
```

#04:00 (tip & trick) Buenas prácticas

Mostrar errores en desarrollo y ocultarlos en producción de forma dinámica

Ver los errores en pantalla por supuesto que es muy útil. Pero ya sabemos que es una muy mala política de seguridad mostrarlos en producción. Alternar entre mostrarlos u ocultarlos es tan simple como definir una constante y cambiar su valor de `true` a `false` dependiendo de si se está en producción o no.

```
const PRODUCTION = false; # true cuando se lleve el código a producción
```

Para que el reporte de errores en desarrollo sea completo, independientemente de cómo esté configurado en el `php.ini`, pueden emplearse las siguientes instrucciones:

```
if(!PRODUCTION) {
    ini_set('error_reporting', E_ALL | E_NOTICE | E_STRICT);
    ini_set('display_errors', '1');
    ini_set('track_errors', 'On');
} else {
    ini_set('display_errors', '0');
}
```

#05:00 (tip) Buenas prácticas

Evitar errores con las rutas de importación al hacer un `include` o `require`

Solo debes establecer la ruta de inclusión a la carpeta raíz de la app e indicar todas las rutas en `includes` y `requires` desde ésta:

```
ini_set('include_path', $_SERVER['DOCUMENT_ROOT']);
```

#06:00 (hack) Soluciones rápidas

Completar una contraseña de forma aleatoria hasta alcanzar la longitud esperada

¿El usuario dejó la contraseña en blanco? ¿utilizó pocos caracteres? En una sola instrucción, la modificamos, generando una clave aleatoria segura y sin hacer absolutamente ningún esfuerzo de validación. Solo hay que emplear funciones anónimas y variables por referencia.

Generación de contraseñas aleatorias seguras con una sola línea de código:

```
$rnd = function(&$a, $n=8) { while(strlen($a) < $n) $a .= chr(rand(33, 126)); };
```

Ejemplo de uso:

```
$rnd($_POST['clave']);      # Modifica el valor de $_POST['clave']  
$rnd($_POST['clave'], 24); # Clave de 24 caracteres
```

El *hack* en acción:

```
php > $rnd = function(&$a, $n=8) { while(strlen($a) < $n) $a .= chr(rand(33, 126)); };  
php >  
php > $rnd($_POST['clave']);  
php > print $_POST['clave'];  
&PSb[Y]K  
php >  
php > $rnd($_POST['clave'], 12);  
php > print $_POST['clave'];  
&PSb[Y]K9sCr  
php > $rnd($_POST['clave'], 32);  
php > print $_POST['clave'];  
&PSb[Y]K9sCr!h=QfQR]iUB3%QP`W:x]
```

Importante: no olvides *hashear* las contraseñas antes de guardarlas:

```
md5($_POST['clave']);
```

#07:00 (truco) Soluciones rápidas

Todos los campos del array `$_POST` a variables, limpios y saneados en un solo paso

Con la ayuda de `array_map` podemos recorrer un array aplicando una misma función a cada uno de sus elementos. Con este sencillo truco, en a penas un paso, limpiamos los campos de un formulario para luego extraerlos en variables:

```
function clean($str) { return htmlentities(strip_tags($str), ENT_QUOTES); }
extract(array_map('clean', $_POST));
```

Ejemplo en funcionamiento:

```
php > $_POST['a1'] = '<b>hola</b> mundo';
php > $_POST['a2'] = 'mundo <script>alert();</script>';
php > $_POST['a3'] = '"hola\' mundo"';
php > function clean($str) { return htmlentities(strip_tags($str), ENT_QUOTES); }
php > extract(array_map('clean', $_POST));
php > print $a1;
hola mundo
php > print $a2;
mundo alert();
php > print $a3;
"holá&#039; mundo&quot;
```

#08:00 (truco) Soluciones rápidas

Importar todos los archivos de una carpeta

Si eres de los que coloca los archivos PHP organizados dentro de directorios, un buen truco para incluirlos de forma automática y despreocuparte de agregar nuevas instrucciones `include` o `require` es el siguiente:

```
$c = "/path/to/folder/";
foreach(scandir($c) as $f) if(is_file($c.$f)) include_once $c.$f;
```

#09:00 (tip, trick & hack) Experimentales

Wrappers y decoradores como en Python

El siguiente código, emula un decorador de restricción de acceso en Python:

```
$decorator = function($func) {  
    $wrapper = function() use ($func) {  
        $logueado = isset($_SESSION['login']) ? $_SESSION['login'] : False;  
        if($logueado) {  
            return call_user_func_array($func, func_get_args($func));  
        } else {  
            return "Debe loguearse";  
        }  
    };  
    return $wrapper;  
};
```

Dada una función candidata a ser «decorada»:

```
function foo($arg1, $arg2...) {  
    # código de la función  
}
```

El **hack** para lograrlo, consiste en retornarla desde un *closure* envuelto a la vez, por el decorador:

```
$foo = $decorator(  
    function($arg1, $arg2...) {  
        return guardar($arg1, $arg2...);  
    }  
);
```

Finalmente se debe llamar al *closure* en vez de llamar a la función original:

```
$foo(1, array(3, 4)...);
```

Tip: Conviene definir todas las funciones normalmente y colocar el decorador a nivel del *core*. Luego, a toda función a la cual se le quiera restringir el acceso (es decir, requiera que un usuario se encuentre *logueado* en el sistema), se le crea el *closure*. Al momento de

efectuar la llamada de la función, si se la tiene que invocar de forma dinámica y no se sabe si es o no restringida, se puede emplear el siguiente **truco** utilizando «variables variables»:

```
$funcion = 'guardar';

if(isset($$funcion)) { # verifica si hay una variable definida con el nombre de la función
    $$funcion($argumentos); # llama al closure
} else {
    $funcion($argumentos); # llama a la función original normalmente
}
```

Ejemplo en funcionamiento:

```
function modificar($id) {
    // código de la función privada ...
    return "Registro con ID $id se ha actualizado" . chr(10);
}

function eliminar($id) {
    // código de la función privada ...
    return "Registro con ID $id se ha eliminado" . chr(10);
}

function ver($id) {
    // código de la función pública ...
    return "Estoy mostrando los datos del registro $id" . chr(10);
}

# Envolturas
$modificar = $decorator(function($id) { return modificar($id); });
$eliminar = $decorator(function($id) { return eliminar($id); });

# ***** pruebas *****
session_start();

$_SESSION['loggin'] = False;
$funcion = 'modificar';
$arg = 15;

if(isset($$funcion)) {
    $salida = $$funcion($arg);
} else {
    $salida = $funcion($arg);
}
print $salida;
// Debe loguearse

$_SESSION['loggin'] = False;
$funcion = 'ver';
$arg = 15;

if(isset($$funcion)) {
    $salida = $$funcion($arg);
} else {
    $salida = $funcion($arg);
}
```

```
print $salida;
// Estoy mostrando los datos del registro 15

$_SESSION['login'] = True;
$funcion = 'eliminar';
$arg = 15;

if(isset($funcion)) {
    $salida = $$funcion($arg);
} else {
    $salida = $funcion($arg);
}
print $salida;
// Registro con ID 15 se ha eliminado
```

#10:00 (hack) Manipulación de archivos

Mostrar archivos binarios desde directorios no accesibles por el navegador

Cuando se trabaja con sistemas de carga de archivos, una buena política de seguridad es almacenarlos en un directorio que no se encuentre servido (es decir, que no sea accesible por el navegador) puesto que el mismo deberá contar con permisos de escritura.

Cuando los archivos que se cargan son binarios que deben ser mostrados estáticamente desde código HTML -o descargados-, es necesario que el navegador tenga acceso a ellos.

Para mostrarlos (o permitir su descarga) sin tener que habilitar un directorio servido con permisos de escritura, el *hack* consiste en servir los archivos mediante PHP, leyéndolos en modo binario:

```
# Archivo: file.php

$archivo = "/path/to/private/folder/{$_GET['f']}";

if(file_exists($archivo)) {
    $finfo = finfo_open(FILEINFO_MIME_TYPE);
    $mime = finfo_file($finfo, $archivo);
    finfo_close($finfo);
    header("Content-Type: $mime");
    readfile($archivo);
}
```

Luego, en los archivos HTML se podrán invocar de la siguiente forma:

```
<img src='file.php?f=imagen.png'>
<a href='file.php?f=archivo.pdf'>Descargar PDF</a>
```


#11:00 (hack) Manipulación de archivos

Mostrar imágenes contenidas en un archivo ZIP no accesible desde el navegador

Este hack implementa el truco anterior para permitir mostrar en HTML, archivos de imágenes que se encuentran dentro de un ZIP.

```
// archivo: image_show.php

$archivo_zip = "imagenes.zip";

if(isset($_GET['filename'])) {
    $imagen = "zip://$archivo_zip#{$_GET['f']}";
    $finfo = finfo_open(FILEINFO_MIME_TYPE);
    $mime = finfo_file($finfo, $imagen);
    finfo_close($finfo);
    header("Content-Type: $mime");
    readfile($imagen);
} else {
    $zipobj = new ZipArchive;
    $recurso = $zipobj->open($archivo_zip);
    $i = 0;
    while($nombre = $zipobj->getNameIndex($i)) {
        print "<img src='image_show.php?f=$nombre'><br>" . chr(10);
        $i++;
    }
    $zipobj->close();
}
```

#12:00 (truco) Orientación a Objetos

Invocar métodos no estáticos de clases instanciables en el mismo paso, en versiones anteriores a 5.4

El truco consiste en crear funciones con el mismo nombre que las clases, que retornen una instancia de la clase:

```
class Foo {  
    function bar() {  
        print 'soy bar';  
    }  
}  
  
function Foo() { return new Foo(); }
```

#13:00 (hack) Orientación a Objetos

Truco #13 aplicado a todas las clases (incluidas las built-in)

El hack consiste en tomar el truco anterior y utilizando la función `get_declared_classes()`, emplearla para crear una función de forma dinámica, por cada clase instanciable del sistema, incluidas las nativas de PHP:

```
$funcion = "  
function CLASE() {  
    \$$args = (func_num_args()) ? func_get_args() : null;  
    return @ new CLASE(\$$args);  
}  
";  
  
$clases = get_declared_classes();  
  
foreach($clases as $clase) {  
    $alias = str_replace('CLASE', $clase, $funcion);  
    eval($alias);  
}  
  
print DateTime()->format('c');  
// salida: 2014-12-07T16:52:50-03:00
```

#14:00 (hack) Orientación a Objetos

Llamar a cualquier función de PHP con estilo orientado a objetos

Mediante este hack podremos invocar cualquier función nativa de PHP, con el estilo orientado a objetos, gracias al método mágico `__call()`:

```
class PHP {  
  
    function __construct() {  
        $funcs = get_defined_functions();  
        $this->builtin_functions = $funcs['internal'];  
    }  
  
    function __call($func, $args) {  
        if(in_array($func, $this->builtin_functions)) {  
            return call_user_func_array($func, $args);  
        } else {  
            print "PHP no posee una función llamada $func";  
        }  
    }  
  
}  
  
function PHP() { return new PHP(); }
```

Ejemplo de uso:

```
print PHP()->str_replace('Mundo', 'Pepe Grillo', 'Hola Mundo!');  
// salida: Hola Pepe Grillo!  
  
PHP()->print_r(array(1, 2, 3));  
/*  
Salida:  
Array  
(  
    [0] => 1  
    [1] => 2  
    [2] => 3  
)  
*/  
  
PHP()->foo();  
// Salida: PHP no posee una función llamada foo
```

#15:00 (hack) Orientación a Objetos

Recuperación de objetos excluyendo propiedades innecesarias

El *hack* consiste en destruir las propiedades del objeto antes de invocar al método `get()`. Por ejemplo, dada una clase `Foo()` similar a la siguiente:

```
class Foo {
    function __construct() {
        $this->propiedad_1 = '';
        $this->propiedad_2 = '';
        # ...
        $this->propiedad_7 = '';
        $this->propiedad_8 = '';
        # ...
        $this->propiedad_que_necesito = '';
    }
}
```

Eliminar mediante `unset()` las propiedades innecesarias antes de llamar a `get()`:

```
$obj = new Foo();
foreach($obj as $property=>$value) {
    if($property != "propiedad_que_necesito") unset($obj->$property);
}
$obj->get(); # $obj solo tendrá la propiedad "propiedad_que_necesito"
```

Esto permitirá ahorrar una innumerable cantidad de recursos cuando un requerimiento -generalmente gráfico- nos demande recuperar solo unas pocas propiedades (un ejemplo típico de ello, es necesitar una lista visual de todos los usuarios del sistema donde solo se muestre el nombre y una ID).

#16:00 (hack) Orientación a Objetos

Guardar datos que no son propiedades del objeto

Nunca conviene llenar un objeto de propiedades con la sola justificación de que es un dato necesario. Si un dato no es una característica del objeto, no debe ser una propiedad del mismo. Sin embargo, si se lo necesita, se puede emplear la inversa del hack anterior para guardarlo.

La tabla en la DB deberá contar previamente con los campos para dichos datos y antes de llamar al método `save()` del objeto, se crearán las «pseudo propiedades» que serán destruidas luego de haberlas guardado:

```
class Foo {
    function __construct() {
        $this->propiedad = '';
    }
}

$obj = new Foo();
$obj->dato1 = 'valor';
$obj->dato2 = 'valor';
$obj->save();
unset($obj->dato1);
unset($obj->dato2);
```

#17:00 (truco) Inteligencia Artificial

Obtener palabras claves de un bloque de texto

```
$palabras = explode(" ", str_replace(chr(10), ' ', $texto));
$unicas = array_unique($palabras, SORT_STRING);
$signos = array('.', ',', ';', ':', '"');
array_walk($unicas,
    function(&$w) use ($signos) { $w = str_replace($signos, '', $w); }
);
$keywords = array();
foreach($unicas as $palabra) if(strlen($palabra) > 2) $keywords[] = $palabra;
$string_keywords = join(" ", $keywords);print
```

Ejemplo de uso:

```
$texto = "Lorem ipsum ad his scripta blandit partiendo, eum fastidii accumsan euripidis in, eum liber hendrerit an. Qui ut wisi vocibus suscipiantur, quo dicit ridens inciderint id. Quo mundi lobortis reformidans eu, legimus senserit definiebas an eos. Eu sit \"tincidunt\" incorrupte definitionem, vis mutat affert percipit cu, eirmod consecetuer signiferumque eu per. In usu latine equidem dolores: Quo no falli viris intellegam, ut fugit veritus placerat per.
```

```
Ius id vidit volumus mandamus, vide veritus democritum te nec, ei eos debet libris consulatu. No mei ferri graeco dicunt, ad cum veri accommodare. Sed at malis omnesque delicata, usu et iusto: zrril meliore. Dicunt maiorum eloquentiam cum cu, sit summo dolor essent te. Ne quodsi nusquam legendos has, ea dicit voluptua eloquentiam pro, ad sit quas qualisque. Eos vocibus deserunt quaestio ei.";
```

```
// string de palabras clave separadas por coma
```

```
print strtolower($string_keywords);
```

```
/*
```

```
Salida:
```

```
lorem, ipsum, his, scripta, blandit, partiendo, eum, fastidii, accumsan, euripidis, liber, hendrerit, qui, wisi, vocibus, suscipiantur, quo, dicit, ridens, inciderint, quo, mundi, lobortis, reformidans, legimus, senserit, definiebas, eos, sit, tincidunt, incorrupte, definitionem, vis, mutat, affert, percipit, eirmod, consecetuer, signiferumque, per, usu, latine, equidem, dolores, falli, viris, intellegam, fugit, veritus, placerat, ius, vidit, volumus, mandamus, vide, democritum, nec, eos, debet, libris, consulatu, mei, ferri, graeco, dicunt, cum, veri, accommodare, sed, malis, omnesque, delicata, iusto, zrril, meliore, dicunt, maiorum, eloquentiam, summo, dolor, essent, quodsi, nusquam, legendos, has, voluptua, pro, quas, qualisque, eos, deserunt, quaestio
```

```
*/
```

#18:00 (tip) MVC

Tip para manejar un sistema ABM mediante URL amigables implementando un pseudo patrón MVC sin emplear OOP

Crea un archivo .php por cada tabla que debas administrar.

Por ejemplo, si tienes 3 tablas: categorias, productos y pedidos, tendrás 3 archivos: categorias.php, productos.php y pedidos.php

Asigna a cada archivo un *namespace* con el mismo nombre pero sin la extensión .php

Por ejemplo, el *namespace* del archivo productos.php será productos y el de pedidos.php será pedidos.

Unifica el mismo nombre de funciones en todos los archivos

Utiliza siempre los mismos nombres de funciones para cada acción (agregar, guardar, editar, actualizar, eliminar, ver, listar).

Así se vería un archivo llamado productos.php

```
namespace productos;

function agregar() {}      # muestra form para agregar
function guardar() {}     # hace un INSERT en la DB
function editar($id) {}   # muestra form para editar
function actualizar() {}  # hace un UPDATE en la DB
function listar() {}      # hace un SELECT en la DB de todos los registros
function eliminar($id) {} # hace un DELETE en la DB
function ver($id) {}      # hace un SELECT en la DB de un solo registro
```

Escribe una regla de reescritura por cada archivo que tengas, redirigiendo la solicitud a un único archivo llamado abm.php

Suponiendo los tres archivos de los ejemplos anteriores, tu .htaccess debería verse así:

```
RewriteEngine ON
RewriteRule ^categorias abm.php
RewriteRule ^productos abm.php
RewriteRule ^pedidos abm.php
```

Crea un archivo llamado abm.php en la raíz de tu aplicación y *rutea* desde allí todas las solicitudes, de forma segura:

```
# Define un array SÓLO con el nombre de los archivos que podrán llamarse
$archivos = array('productos', 'categorias', 'pedidos');

# Define otro array SÓLO con el nombre de las funciones que podrán llamarse en cada archivo
$funciones = array('agregar', 'guardar', 'editar', 'actualizar', 'listar',
    'eliminar', 'ver');

$archivo = 'productos'; # define un archivo por defecto
$funcion = 'listar';    # define una función x defecto

# Las siguientes líneas se encargarán de ejecutar la función que corresponda
# a la solicitud del usuario
$uri = explode('/', $_SERVER['REQUEST_URI']);
if(@in_array($uri[1], $archivos)) $archivo = $uri[1];
if(@in_array($uri[2], $funciones)) $funcion = $uri[2];
$id = isset($uri[3]) ? (int)$uri[3] : 0;

require_once "{$archivo}.php";
$recurso = "{$archivo}\{$funcion}";
$recurso($id);
```

Ejemplo de accesos y *ruteos*:

```
http://www.example.org/productos/agregar -> productos.php agregar()
http://www.example.org/productos/editar/15 -> productos.php editar(15)
http://www.example.org/productos/editar -> productos.php editar(0)
http://www.example.org/categorias/ver/63 -> categorias.php ver(63)
```

Aviso: para que este *tip* funcione, debes tener habilitado el módulo `rewrite` en Apache. Si tienes acceso por consola al servidor, ejecuta el siguiente comando para habilitarlo:

```
sudo a2enmod rewrite && sudo service apache2 restart
```


#19:00 (hack) Vistas en MVC

Alternar entre menú de usuario y menú de administrador

Tener muchos archivos HTML separados es mejor que embeber código HTML en PHP pero sin embargo, existe una técnica mucho mejor aunque más avanzada, que es tener un único archivo HTML con todos los bloques necesarios. El hack para alternar las vistas, consiste en eliminar del HTML los bloques que no se necesitan dependiendo de determinadas condiciones. El mejor ejemplo es alternar la vista de un menú de administrador con uno de usuario común.

Dada una única plantilla HTML, se identifica el comienzo y final de los dos menús mediante comentarios HTML similares:

```
Archivo: template.html
<!doctype html>
.
.
.
<body>
  <tags...>
  .
  .
  .
  <!--menulogueado-->
    <nav>
      este es el menú del administrador
    </nav>
  <!--menulogueado-->

  <!--menuNOlogueado-->
    <nav>
      este es el menú para los usuarios comunes sin permisos
    </nav>
  <!--menuNOlogueado-->
  .
  .
  .
  <otros>
  <tags...>
```

Dependiendo del valor de una variable de sesión (es_admin), se establecerá el identificador del comentario HTML que diferencia ambos menús:

```
$id = isset($_SESSION['es_admin']) ? '' : 'NO';
```

Finalmente, se procede a eliminar el código que corresponda:

```
print preg_replace("/<!--menu{$id}logueado-->/", '', $html);
```

#20:00 (tip) Bases de datos

Ahorrar recursos reemplazando el uso de bases de datos por archivos

Muchas veces empleamos las denominadas «tablas codificadoras» para satisfacer requerimientos visuales. Por ejemplo, una lista de países en un formulario de contacto que solo será enviado por e-mail, no necesita de una tabla codificadora de países ya que estos datos no van a ser relacionados con otros.

Cuando las tablas codificadoras no se emplean en un contexto relacional (es decir, no enlazan ni son enlazadas con otras tablas) dejan de tener sentido por el gran consumo de recursos que generan las consultas y sus conexiones.

La alternativa «económica» es el uso de archivos de texto plano (NO CONFUNDIR con bases de datos NoSQL).

En un archivo de texto, se coloca una lista de los datos necesarios:

```
// archivo: paises.txt
Argentina
Mexico
Colombia
Venezuela
Chile
Ecuador
...
```

Luego se lo *parsea* en un array y se lo ordena de forma ascendente o descendente según se necesite:

```
# se utiliza sort para orden ascendente, rsort para descendente
$orden = "sort";
$paises = explode(chr(10), file_get_contents('paises.txt'));
$orden($paises);
print_r($paises);

// Array ( [0] => Argentina [1] => Chile [2] => Colombia [3] => Ecuador [4] =>
Mexico [5] => Venezuela )
```

#21:00 (trick & hack) Optimización de Bases de datos

Ordenar resultados de una consulta a base de datos sin usar SQL

Reducir costos a la hora de efectuar consultas a la base de datos, es fundamental en cualquier aplicación. Una forma de ahorrar recursos es utilizar funciones de ordenamiento de arrays en lugar de implementar la cláusula ORDER BY con campos no indexados.

Dada la siguiente consulta:

```
SELECT id, denominacion, precio FROM productos;
```

Y obtenidos los siguientes resultados:

```
$rows = array(
    array("id"=>1, "denominacion"=>"Pantalón largo", "precio"=>699.00),
    array("id"=>5, "denominacion"=>"Pollera", "precio"=>412.15),
    array("id"=>12, "denominacion"=>"Camisa manga corta", "precio"=>898.50),
    array("id"=>15, "denominacion"=>"Camisa manga larga", "precio"=>1200.00),
    array("id"=>16, "denominacion"=>"Remera sin mangas", "precio"=>250.00),
    array("id"=>21, "denominacion"=>"Musculosa", "precio"=>220.00),
    array("id"=>22, "denominacion"=>"Camiseta", "precio"=>300.00)
);
```

Emplear el siguiente **truco** para ordenarlos en **PHP 5.5**:

```
# Ordenar por el campo denominacion de forma ascendente (PHP 5.5)
array_multisort(array_column($rows, 'denominacion'), SORT_ASC, $rows);

# Ordenar por el campo precio de forma descendente (PHP 5.5)
array_multisort(array_column($rows, 'precio'), SORT_DESC, $rows);
```

Utilizar el siguiente **hack** para **PHP 5.3 y 5.4**:

```
foreach($resultados as $row) $productos[] = $row['denominacion'];
array_multisort($productos, SORT_ASC, $resultados);

foreach($resultados as $row) $precios[] = $row['denominacion'];
array_multisort($precios, SORT_DESC, $resultados); # Descendente
```

#22:00 (truco) Optimización de Bases de datos

Filtrar resultados de una consulta a base de datos sin usar SQL

Otra forma de optimizar las consultas a bases de datos, es emplear una función de filtrado de arrays en lugar de utilizar la cláusula WHERE LIKE de SQL.

Dada la siguiente consulta (se evita el WHERE LIKE '%...%'):

```
SELECT id, denominacion, precio FROM productos;
```

Y obtenidos los siguientes resultados:

```
$rows = array(
    array("id"=>1, "denominacion"=>"Pantalón largo", "precio"=>699.00),
    array("id"=>5, "denominacion"=>"Pollera", "precio"=>412.15),
    array("id"=>12, "denominacion"=>"Camisa manga corta", "precio"=>898.50),
    array("id"=>15, "denominacion"=>"Camisa manga larga", "precio"=>1200.00),
    array("id"=>16, "denominacion"=>"Remera sin mangas", "precio"=>250.00),
    array("id"=>21, "denominacion"=>"Musculosa", "precio"=>220.00),
    array("id"=>22, "denominacion"=>"Camiseta", "precio"=>300.00)
);
```

Emplear el siguiente **truco** para filtrar los resultados:

```
function filter($row) {
    return (strpos($row['denominacion'], "manga") !== false);
}

$resultados = array_filter($rows, "filter");

// en negritas, el criterio de búsqueda
```

#23:00 (truco) Bases de datos

Generación dinámica de queries

Armar dinámicamente un *query* para seleccionar todos los registros de una tabla con solo contar con el nombre de la tabla es sencillo:

```
SELECT * FROM $tabla;
```

Sin embargo, no solo es una mala práctica, sino que además, no es muy útil si lo que se desea, es conocer el nombre de los campos y asociarlos a su valor correspondiente. Con este sencillo truco y la ayuda de [MySQLiLayer](#)¹, se pueden crear *queries* dinámicos incluyendo el nombre de los campos y obtener arrays asociativos de todos los registros:

```
function get_rows_from_table($tabla) {
    $sql = "SELECT COLUMN_NAME
           FROM COLUMNS
           WHERE TABLE_SCHEMA = ? AND TABLE_NAME = ?";
    $data = array('ss', "nombre_db", "$tabla");
    $fields = array('column'=>'T');
    $rows = mysqli_run_query($sql, $data, $fields, 'INFORMATION_SCHEMA');
    $campos = array();
    // en PHP 5.5 se puede usar array_column
    foreach($rows as $row) $campos[$row['column']];
    $string_campos = join(', ', $campos);
    $nuevo_query = "SELECT $string_campos FROM ?";
    $data = array("s", "$tabla");
    return mysqli_run_query($nuevo_query, $data, $campos);
}
```

Llamando a `get_rows_from_table('productos')` se ejecutará un *query* como el siguiente:

```
SELECT id, denominacion, precio FROM productos;
```

Y se obtendrá un array de resultados como este:

```
$rows = array(
    array("id"=>1, "denominacion"=>"Pantalón largo", "precio"=>699.00),
    array("id"=>5, "denominacion"=>"Pollera", "precio"=>412.15),
    array("id"=>12, "denominacion"=>"Camisa manga corta", "precio"=>898.50),
    array("id"=>15, "denominacion"=>"Camisa manga larga", "precio"=>1200.00),
);
```

1 https://bazaar.launchpad.net/~eugeniabahit/europioexperimental/rel-3.5/view/head:/core/orm_engine/mysqlilayer.php

Europio Engine

HOWTO

© Copyright 2014

Eugenia Bahit

 Sígueme

Compilado de distribución Libre y Gratuita

www.originalhacker.org



© 2014 Eugenia Bahit

Registrado en [SafeCreative](https://www.safecreative.com) (Nº de Registro: [1412232826294](https://www.safecreative.com/1412232826294))

Bajo los términos de la licencia [GNU FDL](https://www.gnu.org/licenses/fdl.html)

(Free Documentation License)

¿Cómo...

Obtener el listado de todos los objetos de un mismo tipo.....	32
Mostrar listado de una colección de objetos con DataTables.....	32
Mostrar un formulario con estilos Bootstrap.....	32
Filtrar y sanear datos de un formulario de forma automática y segura.....	34
Crear una API REST pública para servir datos en JSON.....	35
Consumir una API RESTful empleando cURL.....	35
Crear un módulo con todos los archivos necesarios.....	36
Crear un nuevo modelo con sus vistas y controladores (incluyendo recursos).....	36
Utilizar una colección de objetos como valor para completar un select, checkbox o grupo de radio buttons.....	37
Obtener los últimos objetos agregados.....	37
Filtrar datos de objetos (u objetos) directamente de la base de datos.....	38
Acceder por URL a un archivo binario no servido.....	38
Mostrar un archivo binario no servido desde un recurso (sin usar FileServer).....	39
Generar un error 403 o 404 forzado.....	39
Crear un objeto empleado el patrón Factory.....	40
Componer la propiedad de un objeto utilizando el patrón Composite.....	40
Efectuar una sustitución estática por diccionario.....	40
Realizar una sustitución estática eliminando el código HTML de un comodín cuyo valor es nulo.....	41
Obtener una porción de código HTML desde una plantilla.....	41
Eliminar una porción de código HTML de una plantilla.....	42
Realizar una sustitución iterativa de una colección de objetos.....	42
Usar EuropioCode para codificar un área de texto y prevenir inyecciones de código malicioso.....	43
Crear relaciones entre objetos.....	43
Restringir el acceso a ciertos recursos.....	46
Permitir solo al propietario de los datos modificar o acceder a los datos de su propiedad.....	46
Crear URLs customizadas y slugs.....	47

Obtener el listado de todos los objetos de un mismo tipo

Componente: CollectorObject | Source: core/orm_engine/objects/collectorobject.php

```
$coleccion = CollectorObject::get('NombreDeLaClase');  
$array_de_objetos = $coleccion->collection;
```

Mostrar listado de una colección de objetos con DataTables

Componente: CollectorViewer (plug-in) | Source: common/plugins/collectorviewer/

```
$boton_ver = true;           # para ocultar botón: false  
$boton_editar = true;       # para ocultar botón: false  
$boton_eliminar = true;     # para ocultar botón: false  
  
$tabla = CollectorViewer($array_de_objetos, 'modulo', 'modelo',  
    $boton_ver, $boton_editar, $boton_eliminar)->get_table();  
print Template('Listado de Objetos')->show($tabla);
```

Mostrar un formulario con estilos Bootstrap

Componente: GUI Generator > WebForm PRO (plugin)
Source: common/plugins/guigenerator/forms/webformpro.php

```
$form = new WebFormPRO();  
$form->action = '/modulo/modelo/recurso';  
$form->method = 'POST'; # Valor por defecto: POST  
  
# Campos de formulario (en orden alfabético)
```



```

$col_md = 12; # clase bootstrap col-md-N donde N es un número entre 1 y 12

# Captcha
$form->add_captcha($name, $label, null, null, $col_md);

# Checkboxes
# Requiere preparar una colección de objetos
# seleccion_x_defecto (la ID del objeto a seleccionar por defecto)
# propiedad: nombre de la propiedad a usar como etiqueta
# (el value siempre es la ID del objeto)
Dict::set_dict_for_webform($coleccion, $propiedad, $valor_x_defecto, 'checked');
$form->add_checkbox($name, $label, $coleccion);

# Campo e-mail (HTML 5)
$extras = "requerid placeholder='user@host.com'";
$form->add_email($name, $label, $value, $extras, $col_md)

# Archivos
$extras = 'multiple=multiple';
$name = 'imagenes[]';
$form->add_file($name, $label, $extras='', $col_md);

# Campo oculto
$form->add_hidden($name, $value);

# Contraseña
$extras = 'requerid placeholder=123456';
$form->add_password($name, $label, $value, $extras, $col_md);

# Radio buttons
# Requiere preparar una colección de objetos
# seleccion_x_defecto (la ID del objeto a seleccionar por defecto)
# propiedad: nombre de la propiedad a usar como etiqueta
# (el value siempre es la ID del objeto)
Dict::set_dict_for_webform($coleccion, $propiedad, $valor_x_defecto, 'checked');
$form->add_radio($name, $label, $coleccion);

# Select
# Requiere preparar una colección de objetos
# seleccion_x_defecto (la ID del objeto a seleccionar por defecto)
# propiedad: nombre de la propiedad a usar como etiqueta
# (el value siempre es la ID del objeto)
Dict::set_dict_for_webform($coleccion, $property, $valor_x_defecto, 'selected');
$extras = 'multiple=multiple';
$form->add_select($name, $label, $coleccion, $extras, $col_md);

# Botón «Enviar»
$form->add_submit('Guardar datos');

# Campo de texto
$extras = "style='display: none;'";
$form->add_text($name, $label, $value, $extras, $col_md);

# Área de texto
$extras = "style='height: 12em;'";
$form->add_textarea($name, $label, $value, $extras, $col_md);

# Campo URL (HTML 5)
$form->add_url($name, $label, $value, $extras, $col_md);

# Agrupar campos en un Fieldset
$form->open_fieldset($legend); # Abrir fieldset
# insertar campos dentro del fieldset con $form->add_TIPOCAMPO
$form->close_fieldset();      # Cerrar fieldset

```

```

# Gadgets opcionales
$form->add_title('Título del formulario');
$form->add_instructions('párrafos con instrucciones');

# Barra de navegación
$links = array(
    array('anchor'=>'Link 1', 'href'=>'/path/to/link1'),
    array('anchor'=>'Link 2', 'href'=>'/path/to/link2'),
    array('anchor'=>'Link 3', 'href'=>'/path/to/link3'),
);
$form->add_navbar($links);

# Un enlace solo
$col_md = 12; # clase bootstrap col-md-N donde N es un número entre 1 y 12
$form->addlink('Anchor Text', 'http://example.org', null, $col_md);

# Código HTML personalizado
$form->fields[] = $html; # se agrega en el lugar donde quiere visualizarse

# Zona de errores
# Habilita la clase has-error a los campos con error
# y una caja de alert (alert alert-danger) al comienzo del formulario
$errores = array(
    'nombre_campo_con_error'=>'Texto del error',
    'apellido'=>'El apellido no puede estar vacío',
    'fecha_nacimiento'=>'La fecha de nacimiento es obligatoria'
);
$titulo_alerta = 'Hay algunos errores'; # valor por defecto: Warning!
$form->add_errorzone($errores, $titulo_alerta);

# Generar el formulario
$form->get_form();

# Obtener HTML del formulario
$html = $form->form;

# Imprimir Formulario en pantalla
print Template('Titulo de la página')->show($html);

```

Filtrar y sanear datos de un formulario de forma automática y segura

Componente: SecurityLayer (plugin) | Source: common/plugins/securitylayer/

En el archivo config.ini se debe habilitar la capa de seguridad editando la constante

```
SECURITY_LAYER_ENGINE = '0n'
```

Se recomienda no modificar las otras constantes y dejarlas con el valor por defecto.

Crear una API REST pública para servir datos en JSON

Componente: ApiRESTFul
Source: core/api/server.php

Se debe habilitar la API modificando la constante API_ENABLED en el config.ini

```
API_ENABLED = true
```

En el controlador, crear el recurso (se accederá mediante la URL: /api/modulo/modelo/recurso) almacenando el array u objeto a retornar como JSON (no convertirlo, lo hace la API) en la propiedad \$this->apidata:

```
public function get_all() {  
    $coleccion = CollectorObject::get('NombreDeLaClase');  
    $this->apidata = $coleccion->collection;  
}
```

Consumir una API RESTFul empleando cURL

Componente: APIClient
Source: core/api/client.php

Si la extensión php5-curl no se encuentra instalada, intentará obtener el JSON mediante file_get_contents

```
$json = APIClient::get('http://url-de-la-api.algo');  
$array_u_objeto = json_decode($json);
```

Crear un módulo con todos los archivos necesarios

Componente: Europio CLI (Command Line Interface)
Source: core/cli/ (desarrollado en GNU Bash)

Por línea de comandos, ejecutar:

```
cd core/cli  
./europio -c nombre_del_nuevo_modulo
```

Crear un nuevo modelo con sus vistas y controladores (incluyendo recursos)

Componente: Europio CLI (Command Line Interface)
Source: core/cli/ (desarrollado en GNU Bash)

Por línea de comandos, ejecutar:

```
cd core/cli  
./europio -f nombre_del_modulo NombreDeLaNuevaClaseConCamelCase
```

Luego, editar los archivos de las carpetas models, views y controllers para agregar la información necesaria. Recordar mapear el nuevo modelo y crear la tabla en la DB.

Utilizar una colección de objetos como valor para completar un select, checkbox o grupo de radio buttons

Componente: Dict

Source: core/helpers/dict.php

Se necesita modificar la colección de objetos (array) por referencia y luego emplearla como se indica en el [HowTo](#) de formularios:

```
Dict::set_dict_for_webform(
    $coleccion_de_objetos, # Obtenida con CollectorObject
    $nombre_de_la_propiedad_visible,
    $valor_seleccionado_x_defecto, # siempre será una ID del objeto
    'checked' # si es para un select, indicar 'selected'
);
```

El valor de `$coleccion_de_objetos` es modificado por referencia y por lo tanto no es necesario almacenar ningún valor de retorno en variables.

Obtener los últimos objetos agregados

Componente: DataHandler

Source: core/data/datahandler.php

DataHandler permite la manipulación de objetos como datos.

```
$cantidad = 10; # cantidad de últimos objetos/registros a obtener
$tabla = 'nombredelatabla';
$datos = DataHandler($tabla)->get_latest($cantidad);

# Si se quiere recuperar como un array de objetos (y no de datos)
$objetos = DataHandler($tabla, DH_FORMAT_OBJECT)->get_latest($cantidad);
```

Filtrar datos de objetos (u objetos) directamente de la base de datos

Componente: DataHandler
Source: core/data/datahandler.php

Se debe definir el contenido del «WHERE» del SQL. Por el momento solo admite como operadores de comparación: < (menor que), > (mayor que), = (igual que) y <> (distinto que) y aplica a un único campo:

```
$tabla = 'nombredelatabla';  
$condicion = 'campo<valor';  
$tipo_operador = DH_FILTER_LT;  
DataHandler($tabla)->filter($condicion, $tipo_operador);
```

Operadores posibles:

```
DH_FILTER_EQ          =  
DH_FILTER_NOTEQ      <>  
DH_FILTER_LT         <  
DH_FILTER_GT         >
```

Para obtener los datos como un array de objetos, emplear DH_FORMAT_OBJECT como segunda parámetro de DataHandler:

```
$arr = DataHandler($tabla, DH_FORMAT_OBJECT)->filter($condicion, $operador);
```

Acceder por URL a un archivo binario no servido

Componente: StaticFileServer
Source: core/helper/files.php

La URL será `http://<hostname>/uploadfiles/<archivo>` donde <archivo> será la ruta del binario dentro del path indicado en la constante WRITABLE_DIR del config.ini. Por ejemplo, si se quiere mostrar una imagen llamada 312.png, en HTML se llamará:

```
<img src='/uploadfiles/312.png'>
```

Mostrar un archivo binario no servido desde un recurso (sin usar FileServer)

Componente: FileManager
Source: core/helper/files.php

```
FileManager::show('/path/to/archivo-binario');
```

Si se quiere forzar la descarga, debe pasarse TRUE como segundo parámetro y opcionalmente, como tercer parámetro, puede indicarse el nombre de archivo con el que se desea ofrecer la descarga:

```
FileManager::show('/path/to/foo-702.bar.pdf', True, 'foo.pdf');
```

Generar un error 403 o 404 forzado

Componente: HTTPHelper
Source: core/helper/http.php

```
HTTPHelper::error_response(); # 404 Not Found  
HTTPHelper::exit_by_forbiden(); # 403 Forbiden
```

Crear un objeto empleado el patrón Factory

Componente: Pattern::factory()
Source: core/helper/patterns.php

```
$obj = Pattern::factory('NombreDeLaClase', $id_del_objeto);
```

Componer la propiedad de un objeto utilizando el patrón Composite

Componente: Pattern::composite()
Source: core/helper/patterns.php

```
$obj->propiedad_compuesta = Pattern::composite(
    'NombreDeLaClaseCompositora', $objeto_compositor
);
```

Efectuar una sustitución estática por diccionario

Componente: Template
Source: core/helper/template.php

```
$html = file_get_contents('/path/to/archivo.html');
$diccionario = array('comodin1'=>'valor uno', 'comodin2'=>'valor dos');
$render = Template($html)->render($diccionario);
```

En el HTML los comodines a sustituir se indican entre llaves:

```
{comodin1} {comodin2}
```


Realizar una sustitución estática eliminando el código HTML de un comodín cuyo valor es nulo

Componente: Template

Source: core/helper/template.php

```
$html = file_get_contents('/path/to/archivo.html');
$diccionario = array(
    'comodin1'=>'valor uno',
    'a'=>'', # eliminará el código HTML indicado entre <!--a--> y <!--a-->
    'comodin2'=>'otro valor'
);
$render = Template($html)->render_safe($diccionario);
```

En el HTML, el código que se desea eliminar si el comodín es nulo, debe encerrarse entre comentarios HTML con el nombre del comodín. Ejemplo:

```
<p>{comodin1}</p>
<!--a--><p>{a}</p><!--a-->
<p>{comodin2}</p>
```

Obtener una porción de código HTML desde una plantilla

Componente: Template

Source: core/helper/template.php

```
$html = file_get_contents('/path/to/archivo.html');
$codigo = Template($html)->get_substr('menu-administrador');
```

El código HTML que se obtendrá, estará identificado por comentarios HTML. Ejemplo:

```
<p>Esto no se obtiene</p>
<!--menu-administrador--><nav>esto se obtiene</nav><!--menu-administrador-->
<p>Esto tampoco se obtiene</p>
```

Eliminar una porción de código HTML de una plantilla

Componente: Template
Source: core/helper/template.php

```
$html = file_get_contents('/path/to/archivo.html');  
$html_limpio = Template($html)->delete('menu-administrador');
```

El código HTML que se eliminará, estará identificado por comentarios HTML. Ejemplo:

```
<p>Esto se mantendrá</p>  
<!--menu-administrador--><nav>esto se eliminará</nav><!--menu-administrador-->  
<p>Esto también se mantendrá</p>
```

Realizar una sustitución iterativa de una colección de objetos

Componente: Template
Source: core/helper/template.php

```
$html = file_get_contents('/path/to/archivo.html');  
$render = Template($html)->render_regex('productos', $coleccion);
```

El código HTML sobre el cual se realizará la iteración, estará identificado por comentarios HTML. Ejemplo:

```
<h3>Listado de productos</h3>  
<table>  
<!--productos-->  
<tr>  
<td>{denominacion}</td>  
<td>${precio}.-</td>  
</tr>  
<!--productos-->  
</table>  
<footer>.....</footer>
```

Usar EuropioCode para codificar un área de texto y prevenir inyecciones de código malicioso

Componente: EuropioCode
Source: core/libs/europio_code.php

En la plantilla HTML se debe agregar una instrucción JavaScript para codificar el campos antes de enviar el formulario. Para ello, una vez generado el formulario HTML, si se lo generó con el antiguo plugin WebForm o con el actual plugin WebFormPRO, se debe observar el código fuente para obtener la ID del textarea que se desea codificar. El código JavaScript a incorporar en la plantilla HTML será el siguiente:

```
<script>
  id_del_campo = 'mensaje_3';
  window.onload = function() {
    europio_code = new EuropioCode();
    document.getElementsByTagName('form')[0].onsubmit = function() {
      europio_code.encode(id_del_campo);
    };
  }
</script>
```

Al recibir en PHP los datos del formulario, se puede decodificar mediante la siguiente instrucción:

```
$mensaje = EuropioCode::decode($_POST['mensaje']);
```

Crear relaciones entre objetos

Componente: StandardObject, ComposerObject, MultiplierObject, LogicalConnector
Source: core/orm_engine/objects/

Según el tipo de relación que se necesite generar...

Relación de pertenencia «muchos a uno»

Un objeto compone de forma exclusiva a otro objeto que a la vez es compuesto por varios del mismo tipo: Objeto A (se compone de) Objeto B1, Objeto B2, Objeto B3... Objeto B7

Objeto A debe heredar de StandardObject, tener una propiedad colectora llamada como su compositor, seguida de _collection con valor inicial array():

```
class ObjetoA extends StandardObject {  
    function __construct() {  
        $this->objetob_collection = array();  
    }  
}
```

Objeto B debe heredar de ComposerObject y contar con una propiedad de pertenencia llamada como el objeto compuesto, con valor inicial 0 (cero). El valor de esta propiedad, luego, será la ID del compuesto:

```
class ObjetoA extends ComposerObject {  
    function __construct() {  
        $this->objetoa = 1;  
    }  
}
```

Relación lógica de «muchos a muchos»

Varios objetos de un mismo tipo pueden componer a varios objetos de otro tipo. Objeto A1 (se compone de) Objeto B1 y Objeto B2; Objeto A15 (se compone de) Objeto B2, Objeto B5 y Objeto B6; etc.

Ambos objetos heredan de StandardObject normalmente, pero se requiere crear una tabla de conexiones lógica, ejecutando el CLI:

```
cd core/cli  
./europio -t
```

Siga los pasos eligiendo la opción «c» *conector lógico relacional*.

Para guardar la relación, primero guarde el compuesto, luego agregue uno a uno los compositores empleando el método mágico

```
$objeto_compuesto->add_nombre_del_compositor($compositor)
```

y proceda a guardar, finalmente, la relación lógica:

```

# En el controlador de ObjetoA
$this->model->propiedad = 'foo';
$this->model->save(); # guarda ObjetoA

# Agrega objetos B a objeto A
$this->model->add_objeto_b(Pattern::factory('objetob', 2));
$this->model->add_objeto_b(Pattern::factory('objetob', 5));
$this->model->add_objeto_b(Pattern::factory('objetob', 6));

# Guarda la relación
$lc = new LogicalConnector($this->model, 'ObjetoB');
$lc->save();

```

Relación cíclica de «muchos a muchos» con multiplicación del compositor

El mismo caso que el anterior, pero los Objetos B (con misma ID) se multiplican (como en un pedido donde se solicitan 5 productos con ID n1 y 12 productos con ID n2).

Ambos objetos heredan de StandardObject igual que antes, pero esta vez, se requiere crear una tabla de conexiones lógicas cíclicas, ejecutando el CLI:

```

cd core/cli
./europio -t

```

Siga los pasos eligiendo la opción «m» *Relacional Multiplicador*.

Para guardar la relación, primero guarde el compuesto, luego agregue uno a uno los compositores empleando el objeto relacional MultiplierObject:

```

# En el controlador de ObjetoA
$this->model->propiedad = 'foo';
$this->model->save(); # guarda ObjetoA

# Agrega objetos B a través del multiplicador
$mo = new MultiplierObject($this->model, 'ObjetoA', $objetob_1);
$mo->rel = 5; # cantidad de veces que el mismo objeto B1 compone a A
$mo->save();

$mo = new MultiplierObject($this->model, 'ObjetoA', $objetob_2);
$mo->rel = 3; # cantidad de veces que el mismo objeto B2 compone a A
$mo->save();

```

Para recuperar la relación, deberá sobrescribirse el método get del objeto compuesto:

```

public function get() {
    parent::get();
    $mo = new MultiplierObject($this, 'ObjetoA');
    $mo->get('ObjetoB');
}

```

Restringir el acceso a ciertos recursos

Componente: SessionHandler
Source: core/sessions/

Puede utilizar el módulo users que viene por defecto con Europio Engine para crear nuevos usuarios. El permiso con nivel 1 está reservado para administradores (que siempre tendrán acceso a todo). Puede crear permisos escalados o estrictos.

Permisos escalados: un usuario con nivel 5 puede acceder a funcionalidades que requieran de un nivel 5 o mayor, pero no puede acceder a funcionalidades que requieran un número menor (cuanto más bajo es el número más «autoridad» se tiene). Deberá establecer SESSION_STRICT_LEVEL como false en el config.ini

Permisos estrictos: los usuarios con un nivel determinado solo pueden acceder a funcionalidades con dicho nivel pero no a una que requiera un nivel distinto al que posee. Deberá establecer SESSION_STRICT_LEVEL como true en el config.ini

Para restringir un recurso, entonces, simplemente agregue la siguiente instrucción como primera línea del recurso:

```
@SessionHandler()->check_state($nivel);
```

Permitir solo al propietario de los datos modificar o acceder a los datos de su propiedad

Componente: SessionHandler
Source: core/sessions/

```
@SessionHandler()->check_state($nivel, $id_habilitada);
```

\$id_habilitada será contrastada con la ID almacenada en la sesión.

Crear URLs customizadas y slugs

Componente: Motor MVC

Source: core/mvc_engine/apphandler.php

Debe agregar un archivo llamado `urls.php` en la raíz de su aplicación y en él, definir un array asociativo llamado `$urls` donde las claves sean expresiones regulares que definan la URL y el valor, la URI del recurso real que debe servirse. Este archivo no lleva tags de apertura ni de cierre de PHP ya que es procesado con `eval()`. El *slug* será enviado por parámetro al recurso. Algunos ejemplos:

```
$url = array(
    '^\/$'=>'/modulo/modelo/recurso', # define la URI a mostrar en la home
                                     # para que sea válida, debe establecer
                                     # DEFAULT_VIEW a / en el config.ini
    '^\/productos'=>'/ecommerce/producto/listar',
    '^\/admin$'=>'/ecommerce/pedido/listar'
);
```

¡Feliz 2015!

Por un año más a puro Hacking :)

Eugenia Bahit

 [Sígueme](#)

new WhiteHat();

Cursos de Programación a Distancia

www.cursosdeprogramacionadistancia.com



© 2014 [Eugenia Bahit](#)

Registrado en [SafeCreative](#) (Nº de Registro: [1412232826294](#))

Bajo los términos de la licencia [GNU FDL](#)

(Free Documentation License)